Troubleshooting & Debugging VoIP Calls

Table of Contents

Troubleshoot & Debug VoIP Calls − the Basics

Contents

Introduction

The intent of this document is to demonstrate basic techniques and commands to troubleshoot and debug VoIP networks. An overview of the Voice Call Flow and Telephony Architecture in a Cisco Router is presented, followed by a step−by−step VoIP troubleshooting approach presented in the following steps:

1. Verify Digital and Analog Signalling
2. Verify Digits Received and Sent from the Analog and Digital Voice Ports.
3. Verify End−to−End VoIP Signalling
4. Understand VoIP Quality of Service (QoS) Problems
5. Understand Details of Cause Codes and Debug Values

Note: This document is does not explains every facet of the IOS architecture used in Cisco VoIP gateways and gatekeepers. Rather, it is intended to show which commands can be used and which fields from the command outputs can be most valuable.

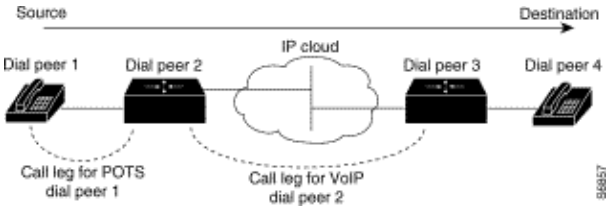
 **Caution**: Debugging the Cisco IOS can be processor intensive, exercise caution when using the

debugs listed below. For more information refer to Important Information on Debug Commands .

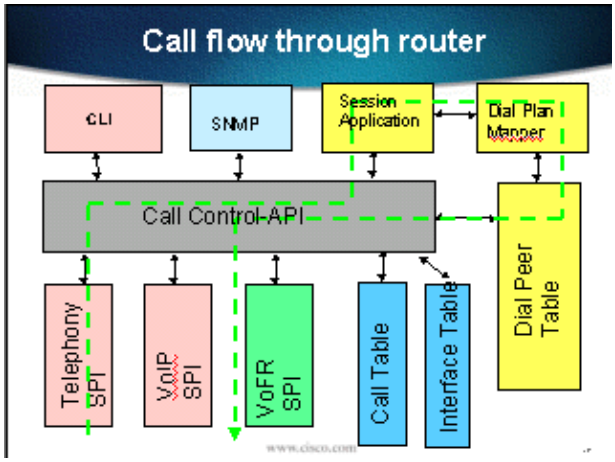
Debugs should be run with timestamps enabled in the log. Enable timestamping by adding the commands: **service timestamps debug datetime msec, service timestamps log datetime msec** in enable mode. The timestaps help determine the interval of time between state changes.

Call Flow in the Network

An important factor to consider before starting any VoIP troubleshooting or debugging is that VoIP calls are made up of three call legs: source POTS (Plain Old Telephone Systems), VoIP (Voice over IP), and destination POTS. This is shown in the diagram below. Troubleshooting & debugging should focus first on each leg independently and then on the VoIP call as a whole.



Router Call Flow



The following definitions explain the function of the main components displayed in router call flow diagram:

Call Control API (Application Programming Interface) –Three clients make use of the call control API: CLI (command line interface), SNMP (simple network management protocol) agent and the Session Application. The Call Control API's (also refered as CCAPI) main functions are to:

- Identify the call legs (i.e. which dial peer is it? where did it come from?)
- Decide which session application takes the call (i.e. who handles it?)
- Invoke the packet handler
- Conference the call legs together
- Start recording call statistics

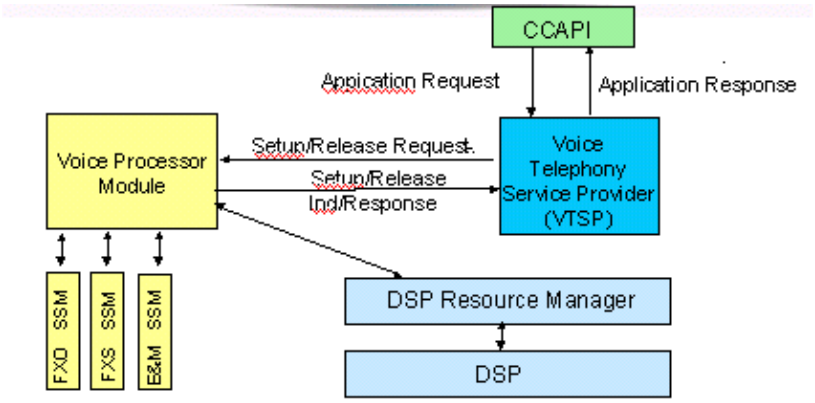
Session Application & Dial Plan Mapper –The Session Application uses the Dial Plan Mapper to map a number to a dial−peer (local POTS or remote VoIP). The Dial Plan Mapper uses the Dial Peer Table to find

active dial−peers.

Telephony & VoIP SPI (Service Provider Interface) –The Telephony SPI communicates with the POTS (analog: fxs, fxo,e&m. Digital: isdn, qsig, e&m, etc) dial−peers. The VoIP SPI is the specific interface to the VoIP peers. Telephony/DSP drivers deliver services to the Telephony SPI while the VoIP SPI relies on session protocols.

Telephony Interface Architecture

This diagram shows the architecture of Cisco router telephony building blocks and how they interact with each other.



The following list describes the functions and definitions of the main diagram components:

- **CCAPI (Call Control Application Programming Interface) –** Software entity that establishes, terminates and bridges call legs.
- **VTSP (Voice Telephony Service Provider)** – An IOS process that services requests from the Call Control API and formulates the appropriate requests to the DSPs (digital signal processor) or the VPM.
- **VPM (Voice Processor Module)** –The VPM is in charge of bridging and coordinating signaling processes between the telephony ports SSM (signaling state machine), the DSP Resource Manager and the VTSP.
- **DSP Resource Manager** –The DSPRM provides interfaces by which the VTSP can send and receive messages to and from the DSPs.
- **Packet Handler** – The packet handler forwards packets between the DSPs and the Peer Call legs.
- **Call Peer** –The Call Peer is the opposite call leg. This can be another telephony voice connection (POTS), VoFR, VoATM, or a VoIP connection.

Verify Digital and Analog Signaling (POTS Call−Leg)

The objectives for verifying digital and analog signaling are to:

- Determine that the proper on−hook and off−hook analog or digital signaling is being received.
- Determine that proper E&M, FXO and FXS signaling is configured on both the router and switch (CO or PBX).
- Verify that the DSPs are in digit collection mode.

The commands outlined below can be used to verify the signaling.

show controllers T1 / E1 (digital)

show controller T1 <*mod/port*> –The first command that should be used. It shows if the digital T1 connection between the router and switch (CO or PBX) is up or down and if it is functioning properly. The output of this command will look like this:

```
router# show controller T1 1/0
T1 1/0 is up.
Applique type is Channelized T1
Cablelength is short 133
No alarms detected.
Framing is ESF, Line Code is B8ZS, Clock Source is Line Primary.
Data in current interval (6 seconds elapsed):

        0 Line Code Violations, 0 Path Code Violations
        0 Slip Secs, 0 Fr Loss Secs, 0 Line Err Secs, 0 Degraded Mins
        0 Errored Secs, 0 Bursty Err Secs, 0 Severely Err Secs, 0 Unavail Secs
```

If using E1 use the **show controllers e1** command. For more information refer to:

- T1 Layer 1 Troubleshooting
- T1 Troubleshooting Flowchart
- Troubleshooting Serial Line Problems

show voice port

show voice port <*mod/port*> –Use this command to display the port state and the parameters configured on the voice−port of Cisco voice interface cards (VIC). Like all IOS commands, defaults will not display in **show running−config**, but they do display with this command.

Sample output for an E&M voice port:

```
router# show voice port 1/0:1
recEive and transMit Slot is 1, Sub-unit is 0, Port is 1
Type of VoicePort is E&M
Operation State is DORMANT
Administrative State is UP
No Interface Down Failure
Description is not set
Noise Regeneration is enabled
Non Linear Processing is enabled
Music On Hold Threshold is Set to -38 dBm
In Gain is Set to 0 dB
Out Attenuation is Set to 0 dB
Echo Cancellation is enabled
Echo Cancel Coverage is set to 16 ms
Connection Mode is normal
Connection Number is not set
Initial Time Out is set to 10 s
Interdigit Time Out is set to 10 s
Call-Disconnect Time Out is set to 60 s
Region Tone is set for US

Voice card specific Info Follows:
Out Attenuation is Set to 0 dB
Echo Cancellation is enabled
Echo Cancel Coverage is set to 16 ms
Connection Mode is normal (could be trunk or plar)
Connection Number is not set
Initial Time Out is set to 10 s
Interdigit Time Out is set to 10 s
Call-Disconnect Time Out is set to 60 s
Region Tone is set for US

Voice card specific Info Follows:
Signal Type is wink-start
Operation Type is 2-wire
E&M Type is 1
Dial Type is dtmf
In Seizure is inactive
Out Seizure is inactive
Digit Duration Timing is set to 100 ms

InterDigit Duration Timing is set to 100 ms
Pulse Rate Timing is set to 10 pulses/second
InterDigit Pulse Duration Timing is set to 500 ms
Clear Wait Duration Timing is set to 400 ms
Wink Wait Duration Timing is set to 200 ms
Wink Duration Timing is set to 200 ms
Delay Start Timing is set to 300 ms
Delay Duration Timing is set to 2000 ms
Dial Pulse Min. Delay is set to 140 ms
```

debug vpm (voice processor module)

The following commands are used to debug the VPM Telephony interface:

- **debug vpm signal** –This command is used to collect debug information for signaling events and can be useful in resolving problems with signaling to a PBX.
- **debug vpm spi** – This command traces how the voice port module SPI (service provider interface) interfaces with the call control API. This debug command displays information about how each network indication and application request is handled.
- **debug vpm dsp** –This command displays messages from the DSP on the VPM to the router and can be useful if you suspect the VPM is not functional. It is a simple way to check if the VPM is responding to off−hook indications and to evaluate timing for signaling messages from the interface.
- **debug vpm all** –This EXEC command enables all of the debug vpm commands: **debug vpm spi**, **debug vpm signal**, and **debug vpm dsp**.
- **debug vpm port** – Use this command to limit the debug output to a particular port. For example, the following shows **debug vpm dsp** messages only for port 1/0/0:

```
debug vpm dsp

debug vpm port 1/0/0
```

For more information: VoIP Debug Commands.

Sample Output for `debug vpm signal` Command

```
maui−voip−austin#debug vpm signal

! −− FXS port 1/0/0 goes from "on−hook" to "off−hook" state
htsp_process_event: [1/0/0, 1.2 , 36] fxsls_onhook_offhook htsp_setup_ind
*Mar 10 16:08:55.958: htsp_process_event: [1/0/0, 1.3 , 8]

! −− Sending ringing alert to called phone
*Mar 10 16:09:02.410: htsp_process_event: [1/0/0, 1.3 , 10] htsp_alert_notify
*Mar 10 16:09:03.378: htsp_process_event: [1/0/0, 1.3 , 11]

!−− End of phone call, port goes "on−hook"
*Mar 10 16:09:11.966: htsp_process_event: [1/0/0, 1.3 , 6]
*Mar 10 16:09:17.218: htsp_process_event: [1/0/0, 1.3 , 28] fxsls_offhook_onhook
*Mar 10 16:09:17.370: htsp_process_event: [1/0/0, 1.3 , 41] fxsls_offhook_timer
*Mar 10 16:09:17.382: htsp_process_event: [1/0/0, 1.2 , 7] fxsls_onhook_release
```

If the on−hook and off−hook are not signaling properly, check the following:

- Verify the cabling is correct.
- Verify that both the router and switch (CO or PBX) are properly grounded.
- Verify that both ends of the connection have matching signaling configurations. Mismatched configurations can cause incomplete or one−way signaling.

For more information on E&M troubleshooting refer to: How to Debug Analog EMSignaling

Sample Output for `debug vpm spi` Command

```
maui-voip-austin#debug vpm spi
Voice Port Module Session debugging is enabled

!-- The DSP is put into digit collection mode.
*Mar 10 16:48:55.710: dsp_digit_collect_on: [1/0/0] packet_len=20 channel_id=128
 packet_id=35 min_inter_delay=290 max_inter_delay=3200 mim_make_time=18 max_make
_time=75 min_brake_time=18 max_brake_time=75
```

Verify Digits Received and Sent (POTS Call-Leg)

Once the on-hook and off-hook signaling are verified to be working correctly, the next step in troubleshooting and debugging a VoIP call is to verify the correct digits are being received or sent on the voice-port (digital or analog). A dial-peer will not be matched or the switch (CO or PBX) will not be able to ring the correct station if incomplete or incorrect digits are being sent or received. Some commands that can be used to verify the digits received/sent are:

- show dialplan number –This command is used to show which dial peer is reached when a particular telephone number is dialed.
- debug vtsp session –This command displays information on how each network indication and application request is processed, signaling indications, and DSP control messages.
- debug vtsp dsp –This command displays the digits as they are received by the voice-port.
- debug vtsp all –This command enables the following debug voice telephony service provider (VTSP) commands: **debug vtsp session**, **debug vtsp error**, and **debug vtsp dsp**.

For more information: VoIP Debug Commands.

show dialplan number

show dialplan number <*digit_string*> –This command displays the dial-peer that is matched by a string of digits. If multiple dial-peers can be matched, they will all be shown in the order in which they are matched. The output of this command will look like this:

```
maui-voip-austin#show dialplan number 5000
Macro Exp.: 5000

VoiceOverIpPeer2
        information type = voice,
        tag = 2, destination-pattern = `5000',
        answer-address = `', preference=0,
        group = 2, Admin state is up, Operation state is up,
        incoming called-number = `', connections/maximum = 0/unlimited,
        application associated:
        type = voip, session-target = `ipv4:192.168.10.2',
        technology prefix:
        ip precedence = 5, UDP checksum = disabled,
        session-protocol = cisco, req-qos = best-effort,
        acc-qos = best-effort,
        dtmf-relay = cisco-rtp,
        fax-rate = voice,   payload size =  20 bytes
        codec = g729r8,   payload size =  20 bytes,
        Expect factor = 10, Icpif = 30,signaling-type = cas,
        VAD = enabled, Poor QOV Trap = disabled,
        Connect Time = 25630, Charged Units = 0,
        Successful Calls = 25, Failed Calls = 0,
        Accepted Calls = 25, Refused Calls = 0,
        Last Disconnect Cause is "10  ",
        Last Disconnect Text is "normal call clearing.",
        Last Setup Time = 84427934.
        Matched: 5000   Digits: 4
        Target: ipv4:192.168.10.2
```

debug vtsp dsp

debug vtsp dsp shows the digits as they are received by the voice-port. The following output shows the collection of DTMF digits from the DSP:

```
maui-voip-austin#debug vtsp dsp
Voice telephony call control dsp debugging is on

!-- ACTION: Caller picked up handset and dialed digits 5000.
!-- The DSP detects DTMF digits. Digit 5 was detected with ON time of 130msec.

*Mar 10 17:57:08.505: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_BEGIN: digit=5,
*Mar 10 17:57:08.585: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_OFF: digit=5,
duration=130
*Mar 10 17:57:09.385: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_BEGIN: digit=0
*Mar 10 17:57:09.485: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_OFF: digit=0,
duration=150
*Mar 10 17:57:10.697: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_BEGIN: digit=0
*Mar 10 17:57:10.825: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_OFF: digit=0,
duration=180
*Mar 10 17:57:12.865: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_BEGIN: digit=0
*Mar 10 17:57:12.917: vtsp_process_dsp_message: MSG_TX_DTMF_DIGIT_OFF: digit=0,
duration=100
```

```
maui-voip-austin#debug vtsp session
Voice telephony call control session debugging is on

!-- <some output have been omitted>
!-- ACTION: Caller picked up handset.
!-- The DSP is allocated, jitter buffers, VAD thresholds, and signal levels are set.

*Mar 10 18:14:22.865: dsp_set_playout: [1/0/0 (69)] packet_len=18 channel_id=1 p
acket_id=76 mode=1 initial=60 min=4 max=200 fax_nom=300
*Mar 10 18:14:22.865: dsp_echo_canceller_control: [1/0/0 (69)] packet_len=10 cha
nnel_id=1 packet_id=66 flags=0x0
*Mar 10 18:14:22.865: dsp_set_gains: [1/0/0 (69)] packet_len=12 channel_id=1 pac
ket_id=91 in_gain=0 out_gain=65506
*Mar 10 18:14:22.865: dsp_vad_enable: [1/0/0 (69)] packet_len=10 channel_id=1 pa
cket_id=78 thresh=-38act_setup_ind_ack
*Mar 10 18:14:22.869: dsp_voice_mode: [1/0/0 (69)] packet_len=24 channel_id=1 pa
cket_id=73 coding_type=1 voice_field_size=80 VAD_flag=0 echo_length=64 comfort_n
oise=1 inband_detect=1 digit_relay=2 AGC_flag=0act_setup_ind_ack(): dsp_dtmf_mod
e()act_setup_ind_ack: passthru_mode = 0, no_auto_switchover = 0dsp_dtmf_mode(VTSP_T
ONE_DTMF_MODE)

!-- The DSP is put into "voice mode" and dial-tone is generated.

*Mar 10 18:14:22.873: dsp_cp_tone_on: [1/0/0 (69)] packet_len=30 channel_id=1 pa
cket_id=72 tone_id=4 n_freq=2 freq_of_first=350 freq_of_second=440 amp_of_first=
4000 amp_of_second=4000 direction=1 on_time_first=65535 off_time_first=0 on_time
_second=65535 off_time_second=0
```

If it is determined the digits are not being sent or received properly, then it might be necessary to use either a digit−grabber (test tool) or T1 tester to verify the digits are being sent at the correct frequency and timing interval. If they are being sent "incorrectly" for the switch (CO or PBX), some values on the router or switch (CO or PBX) might need to be adjusted so that they match and can interoperate. These are usually digit duration and inter−digit duration values. Another item to examine if the digits appear to be sent correctly are any number translation tables in the switch (CO or PBX) that may add or remove digits.

Verify End−to−End VoIP Signaling (VOIP Call−Leg)

After verifying that voice−port signaling is working properly and the correct digits have been received, move to the VoIP Call Control troubleshooting & debugging. The following factors explain why call control debugging can become a complex job:

- Cisco VoIP gateways use H.323 signaling to complete calls. H.323 is made up of three layers of call−negotiation and call−establishment: H.225, H.245, and H.323. These protocols use a combination of TCP and UDP to set up and establish a call.
- End−to−End VoIP debugging will show a number of IOS state−machines, and problems with any state−machine can cause a call to fail.
- End−to−End VoIP debugging can be very verbose and create a lot of debug output.

debug voip ccapi inout

The primary command to debug end−to−end VoIP calls is **debug voip ccapi inout**. The output from a call debug is shown below.

```
!−− Action: A VoIP call is originated through the Telephony SPI (pots leg) to
!−− extension 5000.  <some output has been omitted>

maui−voip−austin#debug voip ccapi inout
voip ccAPI function enter/exit debugging is on

!−− Call leg identification, source peer: Call originated from dial−peer 1 pots
!−− (extension 4000)

*Mar 15 22:07:11.959: cc_api_call_setup_ind (vdbPtr=0x81B09EFC, callInfo={called
=, calling=4000, fdest=0 peer_tag=1}, callID=0x81B628F0)

!−− CCAPI invokes the Session Application

*Mar 15 22:07:11.963: cc_process_call_setup_ind (event=0x81B67E44)
handed call to app "SESSION"
*Mar 15 22:07:11.963: sess_appl: ev(23=CC_EV_CALL_SETUP_IND), cid(88), disp(0)

!−− Allocate call leg identifiers "callid = 0x59"

*Mar 15 22:07:11.963: ccCallSetContext (callID=0x58, context=0x81BAF154)
*Mar 15 22:07:11.963: ccCallSetupAck (callID=0x58)

!−− Instruct VTSP to generate dialtone

*Mar 15 22:07:11.963: ccGenerateTone (callID=0x58 tone=8)

!−− VTSP passes digits to CCAPI

*Mar 15 22:07:20.275:cc_api_call_digit_begin(vdbPtr=0x81B09EFC,callID=0x58,digit=5,
flags=0x1, timestamp=0xC2E63BB7, expiration=0x0)
*Mar 15 22:07:20.279: sess_appl: ev(10=CC_EV_CALL_DIGIT_BEGIN), cid(88), disp(0)
*Mar 15 22:07:20.279: ssaTraceSct: cid(88)st(0)oldst(0)cfid(−1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:20.279: ssaIgnore cid(88), st(0),oldst(0), ev(10)
*Mar 15 22:07:20.327: cc_api_call_digit (vdbPtr=0x81B09EFC, callID=0x58, digit=5
, duration=100)
*Mar 15 22:07:20.327: sess_appl: ev(9=CC_EV_CALL_DIGIT), cid(88), disp(0)
*Mar 15 22:07:20.327: ssaTraceSct: cid(88)st(0)oldst(0)cfid(−1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:21.975:cc_api_call_digit_begin(vdbPtr=0x81B09EFC,callID=0x58,digit=0,
flags=0x1, timestamp=0xC2E63BB7, expiration=0x0)
*Mar 15 22:07:21.979: sess_appl: ev(10=CC_EV_CALL_DIGIT_BEGIN), cid(88), disp(0)
*Mar 15 22:07:21.979: ssaTraceSct: cid(88)st(0)oldst(0)cfid(−1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:21.979: ssaIgnore cid(88), st(0),oldst(0), ev(10)
*Mar 15 22:07:22.075: cc_api_call_digit (vdbPtr=0x81B09EFC, callID=0x58, digit=0
, duration=150)
*Mar 15 22:07:22.079: sess_appl: ev(9=CC_EV_CALL_DIGIT), cid(88), disp(0)
*Mar 15 22:07:22.079: ssaTraceSct: cid(88)st(0)oldst(0)cfid(−1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:23.235: cc_api_call_digit_begin (vdbPtr=0x81B09EFC, callID=0x58, d
git=0, flags=0x1, timestamp=0xC2E63BB7, expiration=0x0)
*Mar 15 22:07:23.239: sess_appl: ev(10=CC_EV_CALL_DIGIT_BEGIN), cid(88), disp(0)
*Mar 15 22:07:23.239: ssaTraceSct: cid(88)st(0)oldst(0)cfid(−1)csize(0)in(1)fDes
```

```
t(0)
*Mar 15 22:07:23.239: ssaIgnore cid(88), st(0),oldst(0), ev(10)
*Mar 15 22:07:23.335: cc_api_call_digit (vdbPtr=0x81B09EFC, callID=0x58, digit=0
, duration=150)
*Mar 15 22:07:23.339: sess_appl: ev(9=CC_EV_CALL_DIGIT), cid(88), disp(0)
*Mar 15 22:07:23.339: ssaTraceSct: cid(88)st(0)oldst(0)cfid(-1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:25.147: cc_api_call_digit_begin (vdbPtr=0x81B09EFC, callID=0x58, d
igit=0, flags=0x1, timestamp=0xC2E63BB7, expiration=0x0)
*Mar 15 22:07:25.147: sess_appl: ev(10=CC_EV_CALL_DIGIT_BEGIN), cid(88), disp(0)
*Mar 15 22:07:25.147: ssaTraceSct: cid(88)st(0)oldst(0)cfid(-1)csize(0)in(1)fDes
t(0)
*Mar 15 22:07:25.147: ssaIgnore cid(88), st(0),oldst(0), ev(10)
*Mar 15 22:07:25.255: cc_api_call_digit (vdbPtr=0x81B09EFC, callID=0x58, digit=0
, duration=160)
*Mar 15 22:07:25.259: sess_appl: ev(9=CC_EV_CALL_DIGIT), cid(88), disp(0)
*Mar 15 22:07:25.259: ssaTraceSct: cid(88)st(0)oldst(0)cfid(-1)csize(0)in(1)fDes
t(0)

!-- Matched dial-peer 2 voip. Destination number 5000

*Mar 15 22:07:25.259: ssaSetupPeer cid(88) peer list:tag(2) called number(5000)
*Mar 15 22:07:25.259: ssaSetupPeer cid(88), destPat(5000), matched(4), prefix(),
peer(81C04A10)

!-- Continue to call an interface and start the next call leg

*Mar 15 22:07:25.259: ccCallProceeding (callID=0x58, prog_ind=0x0)
*Mar 15 22:07:25.259: ccCallSetupRequest (Inbound call = 0x58, outbound peer =2,
dest=, params=0x81BAF168 mode=0, *callID=0x81B6DE58)
*Mar 15 22:07:25.259: callingNumber=4000, calledNumber=5000, redirectNumber=

!-- VoIP call setup

*Mar 15 22:07:25.263: ccIFCallSetupRequest: (vdbPtr=0x81A75558, dest=,
callParams={called=5000, calling=4000, fdest=0, voice_peer_tag=2}, mode=0x0)
*Mar 15 22:07:25.263: ccCallSetContext (callID=0x59, context=0x81BAF3E4)
*Mar 15 22:07:25.375: ccCallAlert (callID=0x58, prog_ind=0x8, sig_ind=0x1)

!-- POTS and VOIP call legs are tied together

*Mar 15 22:07:25.375: ccConferenceCreate (confID=0x81B6DEA0, callID1=0x58, callI
D2=0x59, tag=0x0)
*Mar 15 22:07:25.375: cc_api_bridge_done (confID=0x1E, srcIF=0x81B09EFC, srcCall
ID=0x58, dstCallID=0x59, disposition=0, tag=0x0)

!-- Exchange capability bitmasks with remote VoIP gateway
!-- (Codec, VAD, VoIP or FAX, FAX-rate, etc)

*Mar 15 22:07:26.127: cc_api_caps_ind (dstVdbPtr=0x81B09EFC, dstCallId=0x58, src
CallId=0x59,caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x1 codec_bytes=20,
signal_type=0})

!-- Both gateways agree on capabilities
*Mar 15 22:07:26.127: cc_api_caps_ack (dstVdbPtr=0x81B09EFC, dstCallId=0x58, src
CallId=0x59, caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x1 codec_bytes=20,
signal_type=0})
*Mar 15 22:07:26.139: cc_api_caps_ack (dstVdbPtr=0x81A75558, dstCallId=0x59, src
CallId=0x58, caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x1 codec_bytes=20,
signal_type=0})
*Mar 15 22:07:35.259: cc_api_call_digit (vdbPtr=0x81B09EFC, callID=0x58, digit=T
```

```
, duration=0)
*Mar 15 22:07:35.259: sess_appl: ev(9=CC_EV_CALL_DIGIT), cid(88), disp(0)
*Mar 15 22:07:35.259: ssaTraceSct: cid(88)st(4)oldst(3)cfid(30)csize(0)in(1)fDes
t(0)-cid2(89)st2(4)oldst2(1)
*Mar 15 22:07:35.399: cc_api_call_connected(vdbPtr=0x81A75558, callID=0x59)
*Mar 15 22:07:35.399: sess_appl: ev(8=CC_EV_CALL_CONNECTED), cid(89), disp(0)
*Mar 15 22:07:35.399: ssaTraceSct: cid(89)st(4)oldst(1)cfid(30)csize(0)in(0)fDes
t(0)-cid2(88)st2(4)oldst2(4)

!-- VoIP call is connected

*Mar 15 22:07:35.399: ccCallConnect (callID=0x58)

!-- VoIP call is disconnected. Cause = 0x10

*Mar 15 23:29:39.530: ccCallDisconnect (callID=0x5B, cause=0x10 tag=0x0)
```

If the call is failing and the cause appears to be in the VoIP portion of the call setup, you may need to look at the H.225 or H.245 TCP part of the call setup, as opposed to just the UDP portion of the H.323 setup. The commands that can be used to debug the H.225 or H.245 call setup are:

- **debug ip tcp transaction** & **debug ip tcp packet –** These examine the TCP portion of the H.225 and H.245 negotiation. They returns the IP addresses, TCP ports and states of the TCP connections.
- **debug cch323 h225** – This examines the H.225 portion of the call negotiation. Think of this as the Layer1 part of the 3 part H.323 call setup.
- **debug ch323 h245** – This examines the H.245 portion of the call negotiation. Think of this as the Layer2 part of the 3 part H.323 call setup.

Understand VoIP Quality of Service (QoS) Issues

When VoIP calls are properly established, the next step is to verify that the voice quality is good. Although QoS troubleshooting is not covered in this document, the following guidelines should be considered to achieve good voice quality:

- Understand how much bandwidth a VoIP call consumes with each codec, including Layer−2 and IP/UDP/RTP headers. For more information: Voice over IP − Per Call Bandwidth Consumption
- Understand the characteristics of the IP network the calls will travel over. For example, the bandwidth of a frame−relay network at CIR is much different than that above−CIR (or burst), where packets could be dropped or queued in the Frame−Relay cloud. Take care to ensure that delay and jitter are controlled and eliminated as much as possible. One−way transmit delay should not exceed 150ms (per G.114 recommendation).
- Use a queuing technique that allows VoIP traffic to be identified and prioritized.
- When transmitting VoIP over low−speed links, consider using Layer−2 packet fragmentation techniques, such as MLPPP with Link Fragmentation & Interleaving (LFI) on point−to−point links, or FRF.12 on Frame−Relay links. Fragmentation of larger data packets allows less jitter and delay in transmitting VoIP traffic because the VoIP packets can be interleaved onto the link.
- Try using a different codec and try the call with VAD enabled and disabled to possibly narrow down the issue to the DSP, as opposed to the IP network.

With VoIP, the main things to look for when troubleshooting QoS issues are dropped packets and network bottlenecks that can cause delay and jitter.

Look for:

- interface drops
- buffer drops
- interface congestion
- link congestion

Each interface in the path of the VoIP call should be examined and drops and congestion should be eliminated. Also, round−trip delay should be reduced as much as possible. Pings between the VoIP end points will give an indication of the round trip delay of a link. The round trip delay should not exceed 300ms whenever possible. If the delay does have to exceed this value, efforts should also be taken to ensure this delay is constant, so as not to introduce jitter or variable delay.

Verification should also be made to ensure the IOS queuing mechanism is placing the VoIP packets within the proper queues. IOS commands, such as **show queue <interface>** or **show priority** can assist in verification of queueing.

Details of Cause Codes and Debug Values for VoIP

Use the following tables when reading debugs and the associated values within the debugs.

Q.931 Call Disconnection Causes (cause_codes from debug voip ccapi inout)

Call Disconnection Cause Value (in Hex)	Meaning and Number (in Decimal)
CC_CAUSE_UANUM = 0x1	unassigned number. (1)
CC_CAUSE_NO_ROUTE = 0x3	no route to destination. (3)
CC_CAUSE_NORM = 0x10	normal call clearing. (16)
CC_CAUSE_BUSY = 0x11	user busy. (17)
CC_CAUSE_NORS = 0x12	no user response. (18)
CC_CAUSE_NOAN = 0x13	no user answer. (19)
CC_CAUSE_REJECT = 0x15	call rejected. (21)
CC_CAUSE_INVALID_NUMBER = 0x1C	invalid number. (28)
CC_CAUSE_UNSP = 0x1F	normal, unspecified. (31)
CC_CAUSE_NO_CIRCUIT = 0x22	no circuit. (34)
CC_CAUSE_NO_REQ_CIRCUIT = 0x2C	no requested circuit. (44)
CC_CAUSE_NO_RESOURCE = 0x2F	no resource. (47)
CC_CAUSE_NOSV = 0x3F	service or option not available, or Unspecified. (63)

Codec Negotiation Values (from debug voip ccapi inout)

Negotiation Value	Meaning
1	U−law PCM (G.711u)
2	A−law PCM (G.711a)
3	32k ADPCM (G.726);
4	24k ADPCM (G.726)
5	16k ADPCM (G.726)
6	CS−ACELP (g.729r8 pre−IETF) – high codec complexity
7	CS−ACELP (G.729ar8pre−IETF) – medium codec complexity
8	CS−ACELP (G.729br8) – with IETF VAD
9	G.729abr8
10	G.728
11	G.723r63
12	G.723ar63
13	G.723r53
14	G.723ar53
15	G.729r8
16	G.729ar8

Tone Types

Tone Types	Meaning
CC_TONE_RINGBACK – 0x1	Ring Tone
CC_TONE_FAX – 0x2	Fax Tone
CC_TONE_BUSY – 0x4	Busy Tone
CC_TONE_DIALTONE – 0x8	Dial Tone
CC_TONE_OOS – 0x10	Out of Service Tone
CC_TONE_ADDR_ACK – 0x20	Address Acknowledgement Tone
CC_TONE_DISCONNECT – 0x40	Disconnect Tone
CC_TONE_OFF_HOOK_NOTICE – 0x80	Tone indicating the phone was left off hook
CC_TONE_OFF_HOOK_ALERT – 0x100	A more urgent version of CC_TONE_OFF_HOOK_NOTICE
CC_TONE_CUSTOM – 0x200	Custom Tone − used when specifying a custom tone

CC_TONE_NULL – 0x0	Null Tone

FAX−Rate and VAD Capabilities Values

Values	Meaning
CC_CAP_FAX_NONE 0x1	Fax disables or not available
CC_CAP_FAX_VOICE 0x2	Voice Call
CC_CAP_FAX_144 0x4	14,400 baud
CC_CAP_FAX_96 0x8	9,600 baud
CC_CAP_FAX_72 0x10	7,200 baud
CC_CAP_FAX_48 0x20	4,800 baud
CC_CAP_FAX_24 0x40	2,400 baud
CC_CAP_VAD_OFF 0x1	VAD Disabled
CC_CAP_VAD_ON 0x2	VAD Enabled

Related Information

- **Packet Voice, Video and Telephony**
- **Voice & Telephony Technology Support Page**
- **Voice & Telephony Product Support Page**
- **Access Product Support Page**
- **VoIP Debug Commands**
- **Dial Peer Matching**
- **T1 Layer 1 Troubleshooting**
- **T1 Troubleshooting Flowchart**
- **Troubleshooting Serial Line Problems**